ROBOT CALLIGRAPHY USING PSEUDOSPECTRAL OPTIMAL CONTROL IN CONJUNCTION WITH A NOVEL DYNAMIC BRUSH MODEL

A Dissertation
Presented to
The Academic Faculty

By

Sen Wang

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Engineering
Department of Electrical and Computer Engineering

Georgia Institute of Technology

December 2020

ROBOT CALLIGRAPHY USING PSEUDOSPECTRAL OPTIMAL CONTROL IN CONJUNCTION WITH A NOVEL DYNAMIC BRUSH MODEL

Thesis committee:

Dr. Frank Dellaert
School of Interactive Computing
Georgia Institute of Technology

Dr. Seth Hutchinson
School of Interactive Computing
Georgia Institute of Technology

Dr. Anthony Joseph Yezzi
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Date approved: November 28, 2020

Self-Concept is Destiny

Tal Ben-Shahar

For my family

ACKNOWLEDGMENTS

I would like to thank the members of my thesis committee for their help in the preparation of this work. Professor Frank Dellaert is not only my research advisor who teaches me knowledge and how to do research, but also a life mentor who shows me how to pursue the passion and promote the possibilities of myself. Without his strict requirements and high expectation, I could never finish this excellent work on my own. I also want to express my gratitude to Professor Seth Hutchinson for his great discussion on many motion planning problems during this project. He is also the one who introduces me to the robot art area, which has proven to be my most fortunate experience at Georgia Tech. Finally, I also want to thank Professor Anthony Joseph Yezzi for being a member of my thesis committee and providing help.

Special thanks are due to the friends at Borglab who made this work possible. Jiaqi Chen and Xuanliang Deng are my co-authors for the IROS calligraphy paper who had helped a lot. The most important lab member that I want to thank is Gerry Chen, with whom I have a lot of in-depth discussion about many robotics and optimization problems, and he also helped me with many aspects during the formulation and implementation of the calligraphy project. There are also many excellent students that I am fortunate to work with, and especially, I want to thank Yetong Zhang, Mandy Xie, Abhinav Jain, Shicong Ma, Varun Agrawal, Fan Jiang, Stephanie McCormick, Wanli Qian, Alejandro Escontrela, Carter Price, Ayush Baid, Jing Wu.

Finally, I must express my very profound gratitude to my parents and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Without you, I would never be able to reach this step.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Chinese calligraphy is a unique art form with great artistic value but difficult to master. In this thesis, we formulate the calligraphy writing problem as a trajectory optimization problem, and propose an improved virtual brush model for simulating the real writing process. Our approach is inspired by pseudospectral optimal control in that we parameterize the actuator trajectory for each stroke as a Chebyshev polynomial. The proposed dynamic virtual brush model plays a key role in formulating the objective function to be optimized. Our approach shows excellent performance in drawing aesthetically pleasing characters, and does so much more efficiently than previous work, opening up the possibility to achieve real-time closed-loop control.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Motivation

Chinese calligraphy is one of the most important art forms of traditional Chinese culture. As the combination of both great usefulness and artistic expression, it attracts people's pursue for more than four thousand years [1]. However, training a human to write beautiful calligraphy work is an extremely tough task because of the complexity of Chinese characters and the flexibility of hairy brush [2]. Writing beautiful Chinese calligraphy generally requires more than several decades of' practice.

Making robots write beautiful calligraphy is also difficult, as learning and mastering this art form takes humans years of practice. Chinese characters are complex and a calligraphy brush is difficult to manipulate properly. Hence, making a robot achieve comparable results is a worthwhile endeavor, both to expand the robot's capabilities into art as well as push our understanding of how to manipulate deformable brushes.

1.2 Current issues

Most relevant research in this area adopts either a learning-based method or an approach based on trajectory optimization. The former includes learning from demonstration [3, 4], or learning from visual feedback given a reference image [5]. By using learning, one can eliminate the difficulty of modeling the behavior of a real calligraphy brush. However, learning methods have a large training cost and may not generalize well to previously unseen characters. On the other hand, trajectory optimization methods do not face these problems: they *simulate* the writing behavior of an actual brush, and then search for an optimal trajectory for the robot to execute [6, 7]). The difficulty there is that most simulated

brush models [8, 7] do not account for the complex ways a brush deforms during the writing process. Being able to capture the complexity of a deformable brush has an important influence on the final performance.

1.3 Main contribution

In this paper, we propose a novel virtual brush model intended to capture the dynamics of an actual calligraphy brush, which is an improved version based on an earlier virtual brush model by Kwok [6], which we use as a baseline. We improve on this model by explicitly modeling the brush dynamics, while retaining a much simpler structure compared to the elaborate models in other work [9, 10]. We then use this model to simulate the writing process for a given open-loop trajectory, which allows us to then optimize for the trajectory parameters.

To do so, our second contribution is an efficient trajectory optimization-based method based on *pseudospectral optimal control* [11],[12] that achieves fully automatic writing of Chinese characters given a character's unicode. Pseudospectral methods are based on Legendre or Chebyshev polynomials, which are excellent at approximating and representing continuous trajectories and controls. Our method converges quickly and efficiently even when using many control points.

Finally, we exploit the existence of vector-based character databases such that, given a character's unicode, we immediately have access to the individual character strokes as well as the stroke order. We optimize for each stroke separately, and to obtain even faster convergence we initialize the trajectory for each stroke by a stroke skeleton which is also available in these databases.

Our proposed continuous, nonlinear optimization framework differs from previous work [6, 7, 8] which uses heuristic optimization methods known for slow convergence rates and a high computational cost. Hence, our method can potentially be used in a closed-loop control system, which we intend to pursue in future work.

(a) Simulated image from initial trajectory estimate

(b) Simulated image after trajectory optimization

(c) Robot executing trajectory

(d) Written image

Figure 1.1: Summary of workflow

This paper summarizes and extends materials from my previous publication [13], which also wins the Final list reward of IROS Best Entertainment and Amusement Paper.

CHAPTER 2

RELATED WORK

There is much work in the robotics community that aims to create art using robots, such as painting and drawing [14, 15, 16], sculpture[17], graffiti [18], etc. Below we focus our discussion on robot calligraphy, most algorithms using a calligraphy brush can be categorized as either learning or trajectory optimization-based.

2.1 Calligraphy robots using learning-based methods

Examples of simple learning-based methods include Sun *et al*. 's learning from demonstration [19] [3], Mueller *et al*. 's trial-and-learn iterative learning method [5]. Some more advanced learning algorithms such as RNN [20], generative adversarial networks [21], deep reinforcement learning [22], and local and global learning models [4] were also explored. However, these methods require many iterations of training to achieve good performance, and have difficulty generalizing to new and/or complicated characters.

2.2 Virtual brush models

Trajectory optimization-based algorithms mainly rely on *virtual brush models*, which can be divided into two categories: physics-based models and data-driven models.

Physics-based virtual brush models strive to simulate the physical dynamics of a real brush from experimental observation [23, 24, 25, 26] or physical laws [27, 10]. Strassmann [28] proposes an initial design featuring four basic parameters of a hairy brush based on bristles. Wong *et al*. [29] propose to use a cone to represent the bundle of the brush and use the cross-section of the cone, an ellipse, to represent the footprint of the brush. Xu *et al*. [9] propose a detailed virtual brush model for use in synthetic imagery, with good

results in creating realistic-looking simulations. However, obtaining and fitting good parameters to the models above is difficult. As such, we propose a virtual brush with an easy structure to fit and implement, geared towards real-time trajectory optimization.

Data-driven virtual brush models are created by measuring and recording actual brush footprints on writing surfaces. Kwok *et al*. propose a simple virtual brush which draws droplet shapes whose size is proportional to the writing height [6]. In their later work [8], they use a camera placed below the writing plane to collect footprints during the writing process. Lam *et al*. [7] define their writing mark as a polygon connected by eight points and fit their position parameters with the collected footprints. With an eye for efficiency, Baxter *et al*. [30] build a deformation table to speed up the associated computation while still being able to simulate complex effects.

2.3 Stroke extraction

Stroke extraction involves separating a character into its comprising strokes and is difficult to do with good accuracy when analyzing only the pixels of an image. There are three main categories of stroke extraction methods: skeleton-based [31, 32], region-based [33], and contour-based [34, 35, 36]. Most of these methods are complex and may not achieve good results as compared to the ground-truth stroke segmentation. As such, we propose using vector-based character database, which provides a quick and accurate way to extract strokes, as well as stroke order.

2.4 Optimization methods

As mentioned above, Kwok *et al*. [6, 8] propose using genetic algorithm for direct character-level optimization guided by their brush models. However, their reported computation times are long, and the strokes have to be manually separated. Lam *et al*. [7] minimize the width difference of the strokes between reference images and a simulated image written by the virtual brush as it moves along the middle axis of each stroke. However, their

method is sensitive to small variations in stroke images, and so the results suffer from a loss of smoothness. One of our main contributions below is the use of methods from optimal control to achieve the desired stroke trajectories.

CHAPTER 3
VIRTUAL BRUSH MODELS

The virtual brush model is one of the most important parts of the whole project. Many modern trajectory optimization algorithms could generate good results at very fast speed. However, without the help of an excellent brush model to evaluate the error distribution, there is no way to promise the trajectory found could produce satisfied calligraphy work.

The idea of using virtual brush models originates from the pipeline of model predictive control (MPC), where a prediction model also predicts the system evolution given a control signal. Generally, out of consideration of both lower calculation cost and less accumulation of prediction error, MPC runs optimization over a small time period to decide the control strategy at the current time instance. After it, it shifts one step forward and re-runs the optimization in the new time period. This strategy is also called receding horizon control. In our situation, one stroke is generally not a big step to accumulate big prediction error; furthermore, collecting data and re-running the optimization algorithm may take a longer time, and so make the writing process extremely slow. As such, we simply run an optimization for a single stroke at one time and do not adjust the stroke trajectory during the writing process.

3.1 Philosophy behind virtual brush model design

3.1.1 Modeling brush behaviors

As is known, real soft brushes are composed of many bristles of different lengths in a complicated way. Such composition gives it a lot of freedom in expression but also increases the great difficulty in brush modeling. The same brush model under different usage situations or conditions could show totally different and complicated behavior. For example, when

the brush is pushed down too much, the bristles could split into a different direction in a chaotic way, which is highly difficult to predict accurately. As a second example, when the brush is written following the same command (at a reasonable height) but with a different amount of ink, the tip of the brush could also show different splitting patterns and generate different texture in strokes. Finally, because of the elastic property of the soft tip, after pushing the brush down onto the paper, changing the position of the brush holder slightly may only result in the deformation of the shape of the tip without changing the shape of the written mark. This is caused by the wet friction between the brush and the paper.

However, although the brush model could have very complicated behavior during usage, not all of them appear frequently in the normal writing process. For example, the brush could only split in the "cursive" style. In the "official" and "regular" styles it is generally avoided. Also, the ink is mostly supplied at a reasonable amount in the "official" and "regular" styles as opposed to the "cursive" style. As such, devoting much energy to modeling the 'splitting' and 'texture' effect is not worthwhile if the main purpose is actually about writing "regular" style calligraphy work.

In this project, we mainly consider writing the "regular" style of Chinese calligraphy, whose potential writing behavior includes:

1. Having a unique initial state where no deformation applies
2. Pushing down the brush model vertically
3. Dragging the brush to move following a straight line
4. Dragging the brush to make a turn with an angle
5. Pushing down the brush model while still dragging it to move around
6. Lifting the brush model vertically
7. The brush holder is always perpendicular to the writing paper, although the tip of the brush could have any kind of deformation

In the following, we will collectively call these 7 requirements as *behavior requirements*. These are the requirements that any good virtual brush model should not only provide but also promise accurate simulation results.

3.1.2 Brush parameters

After performing experiments and verification for a long term, we propose a set of core requirements to describe the virtual brush model's state parameters. An ideal collection of the state parameters should satisfy requirements as follows:

1. Rendering the brush's state parameters into paper could produce a highly similar written mark as the real brush
2. One unique real brush's state could only be described by one set of parameters
3. The final picture matrix exhibit continuous, and linear, or at least, locally linear relationship with respect to the parameters. This requirement helps optimization algorithms find global optimization results
4. The rendering process should not be too complicated to avoid long optimization calculation time
5. The number of parameters should not be as few as possible for easier model identification

The first two requirements decide that the virtual brush could serve as a bijective function; the third requirements make the virtual brush work well with a general nonlinear optimization algorithm as illustrated in the following chapter; the last two requirements are posed to achieve low calculation cost for better convenience during usage. In the following, we will collectively call these 5 requirements as *modeling requirements*.

To evaluate a brush model, we can first go through all the behavior requirements to see whether the brush model is able to provide such a simulation effect; after it, we can go

through all the modeling requirements and individually evaluate how well do they satisfy the proposed requirements.

Guided by the requirements stated above, there are two kinds of virtual brush models that we considered and implemented in this paper: a simple virtual brush model and a dynamic virtual brush model. The simple virtual brush model satisfies requirements 3 5 perfectly; however, after performing more experiments and analysis, we decided that the simple virtual brush is not accurate enough to produce advanced calligraphy work, and so we propose the novel dynamic virtual brush model that is able to simulate the motion of the real brushes more accurately with low calculation cost. The dynamic virtual brush model achieves a much better effect than the simple virtual brush model on requirements 1 and 2, though a little worse on requirements 3 5; Overall, we treat it as a necessary compromise for a much better writing effect. Next, we will introduce the two virtual brush models respectively.

3.2 Simple virtual brush model

3.2.1 Dynamics

The simple virtual brush model is similar to Kwok *et al*. 's work [6] and is used as a baseline comparison in our results. The simple brush model only has one parameter to decide its shape, the full set of simple virtual brush's state parameters are given as follows:

$$\mathcal{X}_{simple} = \{r,\ x,\ y,\ z\} \tag{3.1}$$

where r is the radius of the circle which represents the written mark, $x,\ y,\ z$ is the coordinate of the brush holder.

The dynamics mechanism of the simple virtual brush is also pretty simple. The only parameters to describe the shape of the written mark, r, is assumed to follow a linear

relationship with the pushing depth z:

$$r = k \cdot z + b \tag{3.2}$$

The center position of the written mark is always assumed to be at the perpendicular position of the end-effector(which holds the brush holder). Since we always make the robot hold the brush perpendicular to the paper, this assumption actually assumes that the tip of the brush does not have horizontal deformation during the writing process: the center of the written mark is always the same as the center of the end-effector.

Note that we align the task coordinate frame for the robot and brush such that the $z = 0$ plane corresponds to the plane of the paper and z increases as the brush is moved towards the paper.

One easy generalization can be made in the shape of the written mark. Instead of assuming the circle shape, Kwok *et al*. also proposed using more complex templates (such as droplets), but we get that behavior for free by modeling the dynamics in our model below.

3.2.2 Analysis

We check the behavior requirements at first. Although all the required behaviors are supported by the simple virtual brush model, we can easily see that many of them do not behave similarly to the real writing brushes. For example, when making a turn, the simple brush would always make a very smooth round turn, but the real brush usually has a more complicated turning shape depending on how to use it.

As for modeling requirements, we can easily see that requirements 3, 4, and 5 are satisfied perfectly. The written mark, which is represented by a picture matrix, obviously shows an excellent linear relationship with respect to the parameters; the calculation cost, which in this case is simply drawing a circle, is also pretty low; and there is almost no way

to reduce the number of parameters even further. However, requirements 1 and 2 are only roughly satisfied: although we can change the size of the simulated written mark freely, the shape of a real written mark is still pretty different from a circle; and it is also hard to establish a single bijective mapping between a circle and the real written mark, which contains an infinite number of shapes that is similar to a circle.

Obviously, the simple virtual brush model ignores a lot of detail of the brush. Especially, it ignores the deformation of the tip in the writing process and assumes the written mark left on the paper is always a circle. The only factor that influences the shape of the written mark is the pushing depth of the brush. As the end-effector pushes the brush lower, the radius of the written mark becomes bigger following a linear relationship.

3.3 Dynamic virtual brush model

As opposed to the simple virtual brush, a dynamic virtual brush with higher simulation accuracy is generally required for robots writing good calligraphy work. Given a sequence of control commands v_{xi}, v_{yi}, v_{zi}, the dynamic virtual brush updates its state parameter sequences and draws the corresponding written mark. Based on factors that have a dominant influence on the writing results, such as the shape of written mark, brush deformation in the writing process, we design the dynamic virtual brush that could compete for a true brush.

The dynamic virtual brush model has two components: a drawing component, and a dynamic update component. The drawing component describes how the brush leaves a mark on paper depending on its parameters. The updating component then describes how the brush parameters are updated due to deformations when executing an open-loop trajectory $[x(t), y(t), z(t)]^T$.

3.3.1 Drawing component

The dynamic virtual brush model has four state parameters governing its drawing footprint: width w, drag d, offset o, and orientation θ. As shown in Figure 3.1, orientation

θ describes the direction of the writing mark on the paper, width w and drag d define the shape and size of a brush mark, the offset o and orientation θ together simulates the deviation of the brush mark from the center of the vertical brush handle due to bending of the brush hairs when applying pressure. As Figure 3.1 shows, together with the brush holder coordinates $\boldsymbol{X} \doteq (x, y, z)$, these four parameters define the 7-dimensional brush state $\mathcal{X} \doteq (x, y, z, \theta, o, w, d)$.

The shape of the brush mark is parameterized by a quadratic curve which is uniquely determined by the width w and the drag d. Some more specialized brush behavior such as intentional hair-splitting is reserved for more artistic calligraphy styles and is not modeled here. To summarize, given a complete set of parameters $\mathcal{X}_i$, we can render a written mark in simulation as follows:

$$D(\mathcal{X}_i) = I_i \tag{3.3}$$

where I_i is the simulated written mark in the picture matrix form.

Root point

From Figure 3.1, given the state $\mathcal{X}$, our model then computes the "root" location of the brush mark as

$$\boldsymbol{X}^{root} = \pi(\boldsymbol{X}) - o\mathcal{V}(\theta) \tag{3.4}$$

where $\pi(\boldsymbol{X})$ is the projection of $\boldsymbol{X}$ in the $z = 0$ plane and $\mathcal{V}$ transforms the scalar orientation θ into a three-dimensional unit vector in that same plane. There are two important properties associated with the root point:

1. Offset actually describes the horizontal distance between root point and brush holder (overlapped with end-effector coordinate in our case)

2. Given the position of root point, brush holder coordinate, and offset, the orientation can be uniquely decided from Equation 3.4

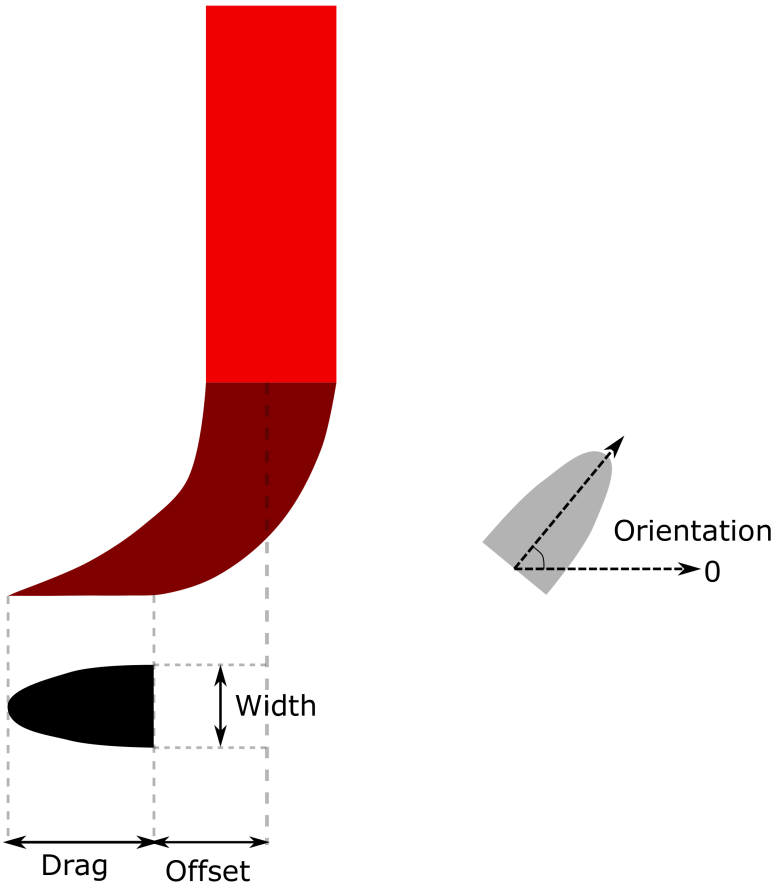


Figure 3.1: Dynamic virtual brush model and its parameters. The root location of the brush mark is defined as the middle of the flat end and is determined by the brush holder position $\boldsymbol{X}$, the offset o, and the orientation θ, whereas the shape of the brush mark is determined by the width w and drag d.

These two important properties are used to derive the updating principals for the offset and orientation parameters in the next section.

3.3.2 Dynamic component

These two important properties are used to derive the updating principles for the offset and orientation parameters in the next section.

$$\mathcal{X}_{i+1} = f(\mathcal{X}_i, \mathcal{U}_i) \tag{3.5}$$

where $\mathcal{U}_i = (v_{xi}, v_{yi}, v_{zi})$ is the control at the i^{th} time instance. The x, y, and z components of the state $\mathcal{X}_0$ at $i = 0$ are initialized from the first point in the open-loop trajectory. Initializing the other components is more involved and is discussed in more detail in the next chapter.

Brush holder coordinates

Updating on the brush holder coordinates is pretty straightforward. Assuming that the end-effector holds the brush with no slippery movement, the position of the end-effector is always aligned with the brush holder. As such, updating on the brush holder coordinates is given as follows:

$$x_{i+1} = x_i + v_{xi}\Delta t \tag{3.6}$$

$$y_{i+1} = y_i + v_{yi}\Delta t \tag{3.7}$$

$$z_{i+1} = z_i + v_{zi}\Delta t \tag{3.8}$$

$$\tag{3.9}$$

3.3.3 Width and drag

There is no straightforward updating rule applied to the abstract parameters width and drag. As such, we did a lot of experiments to explore the possible dependency between these parameters. To be more specific, we control the robots to write strokes given different writing trajectories and then record the shape parameters which are able to describe the written results. Figure 3.2 shows some important findings. The x-axis shows the z parameter during the writing process, which basically shows how much the brush is pushed down. The y-axis shows the corresponding shape parameters recorded by controlling the robot to write strokes under different circumstances. Overall, it is easy to see that width and drag parameters have a good linear relationship with respect to the z parameter. As such, we run linear regression based on collected data, and use the fitting results as the basic updating functions (i.e., the Width and Drag function in the formula Equation 3.10 and Equation 3.11 for the width and drag parameters.

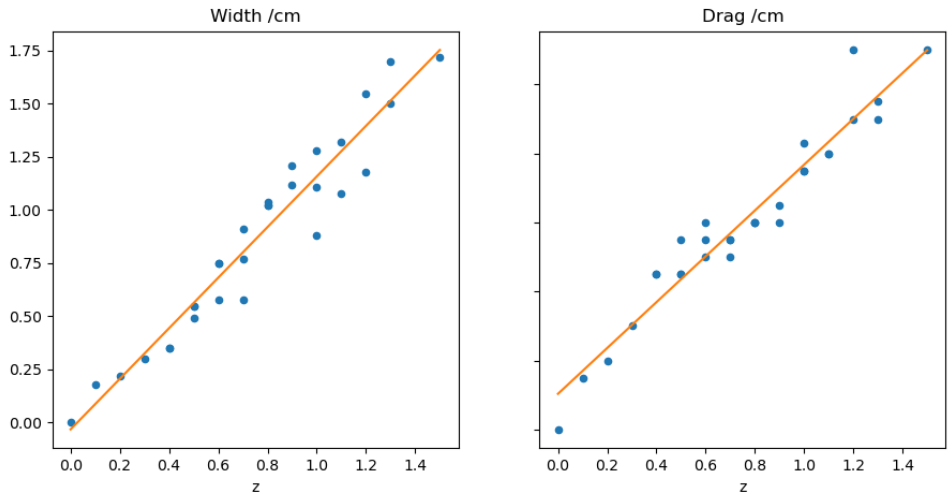


Figure 3.2: The data collected for fitting the model parameters width and drag for our new virtual brush model (2 outliers were removed).

However, we find that simply applying these regression results is not enough. For

example, a sudden change in the z parameter actually does not always introduce a sudden change on the width and drag parameters. Therefore, we also introduce the inertia factor to the updating formulas. We have used 0.02 in our experiments - modeling the fact that the deformations happen gradually and steadily. The final updating formulas are given as follows:

$$w_{i+1} = w_i K_w + \textit{Width}(z_{i+1})(1 - K_w) \tag{3.10}$$

$$d_{i+1} = d_i K_d + \textit{Drag}(z_{i+1})(1 - K_d) \tag{3.11}$$

Offset and orientation

3.3.4 Offset and orientation

The most difficult part of the dynamics model is about how to update offset and orientation properly. After a long time of observation and analysis, we propose the updating principles to approximate the real dynamics in most writing situations.

To explain the motivation for the proposed dynamics, we describe the real writing process at first. When people hold the brush holder and move it for a long distance, the tip of the brush will move and follow correspondingly; however, during the writing process, if people only move the brush holder for a very small distance, it is also pretty possible that the brush holder does not move at all. This is because of the influence of the friction and surface tension between the contact between the wet brush and the paper. When people use the brush to write on paper, these two forces make the tip of the brush stick with the paper, or in other words, have a tendency to stay in the old position even though a new control command may already move the brush holder to a new position. Another way to describe the phenomenon is that the 'offset' parameter is a variable that could take values in a range if only the 'z' parameter is given. Therefore, there is a threshold to decide whether the tip

of the brush moves or not. If the distance between the brush holder and the old position of the tip is too large, the tip moves to follow the brush holder; however, if the distance is small, then the wet friction stops the tip of the brush from moving.

To decide the threshold, we go back to our previous experiments, and use the collected data to estimate an experimental value.

Based on the observation, the updating formulas for the offset parameter is given as follows:

$$o_{i+1} = \min(\textit{Offset}(z_{i+1}),\ \textit{Offset'}(\boldsymbol{X}_i^{root}, \boldsymbol{X}_{i+1})) \tag{3.12}$$

where the $\boldsymbol{X}^{root}$ means the root point introduced in the drawing component and is used there to represent the position of the tip of the brush. At each time instance, we assume the tip does not move after the brush holder coordinates are updated. If the distance between the brush holder and the root point is smaller than the threshold, then our assumption is right, and we update offset as the horizontal distance between the brush holder and the root point. However, if the distance is larger than the threshold, then the tip must have moved, we update the offset parameter directly as the threshold.

The orientation is defined by the displacement between the root point and the horizontal projection of the brush holder coordinate. If the brush tip does not move, then the orientation can be directly updated according to its definition; if the brush tip moves, then the orientation is taken as the orientation between the new horizontal brush holder coordinates and the old root point. The final updating formulas are given as follows:

$$\theta_{i+1} = \mathcal{V}^{-1}(\boldsymbol{X}_{i+1}^{root} - \pi(\boldsymbol{X}_{i+1})) \tag{3.13}$$

3.3.5 Complete dynamic component

To summarize, the whole dynamic model is given as follows

$$
\begin{aligned}
x_{i+1} &= x_i + v_{xi}\Delta t \\
y_{i+1} &= y_i + v_{yi}\Delta t \\
z_{i+1} &= z_i + v_{zi}\Delta t \\
w_{i+1} &= w_i K_w + \textit{Width}(z_{i+1})(1 - K_w) \\
d_{i+1} &= d_i K_d + \textit{Drag}(z_{i+1})(1 - K_d) \\
o_{i+1} &= \min(\textit{Offset}(z_{i+1}),\ \textit{Offset'}(\boldsymbol{X}_i^{root}, \boldsymbol{X}_{i+1})) \\
\theta_{i+1} &= \mathcal{V}^{-1}(\boldsymbol{X}_{i+1}^{root} - \pi(\boldsymbol{X}_{i+1}))
\end{aligned}
\tag{3.14}
$$

3.3.6 Analysis

Now we evaluate the new dynamic virtual brush model based on our previous criterion. The required behaviors are all provided by the dynamic virtual brush model, and the simulation results are also more similar to the real brushes, especially comparing with the simple virtual brush model.

Then we can look at the modeling requirements. Comparing with the simple virtual brush model, requirement 1 gets a big improvement because of the much more careful modeling process. The brush parameters are fitted based on collected real written marks, which make the virtual brush model behave more similarly as the real brush. Requirement 2 is also satisfied better too.

Requirements 3, 4, and 5 are less satisfied than the simple virtual brush model because of the more complicated dynamics mechanism. However, the dynamic model is still not a very complicated one because the simple virtual brush model is simply too simple. The overall calculation is still pretty fast and we can generate good trajectories in a normal laptop for several seconds. The linearity is also kind of violated, but such sacrifice is

necessary for better simulation because the writing brush model itself does not follow a linear relationship at all.

As such, to summarize, although the new virtual brush model is more complicated and slower than the simple brush model, it is expected to work much better and guide excellent written results, and so would be the main virtual brush model to adopt in this thesis.

CHAPTER 4

SYSTEM IDENTIFICATION

After introducing the virtual brush model, in this chapter, we show how to estimate the parameters of a virtual brush model so that it can simulate a real writing brush. This is actually an important step for robot calligraphy. No matter how perfect a virtual brush model is, it has to match the real brush to guide the writing process.

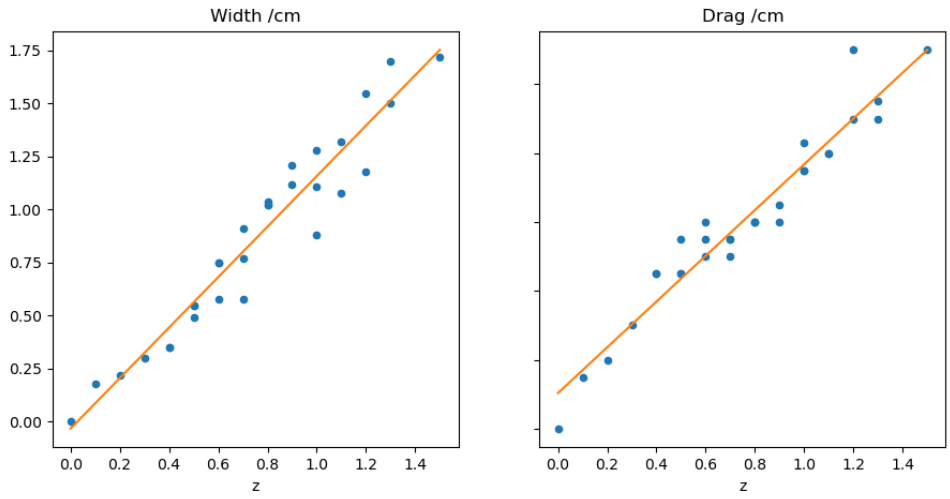


Figure 4.1: The data collected for fitting the model parameters width and drag for our new virtual brush model (2 outliers were removed).

Parameter estimation for the Width function in Equation 3.10, Drag function in Equation 3.11, Offset function in Equation 3.12 can be gotten from Figure 4.1. By controlling the robot to write with different z parameters under different situations, it is not hard to collect enough data to run regression as described above. In our case, the drag parameter is gotten by collecting written mark when we pushed the brush down at different depth; as for width and offset, we control the robot to write straight lines at different depth, which we believe would generate more reliable measurements. After it, we take the width of lines for the width parameter, and the horizontal difference between the ending point of the written

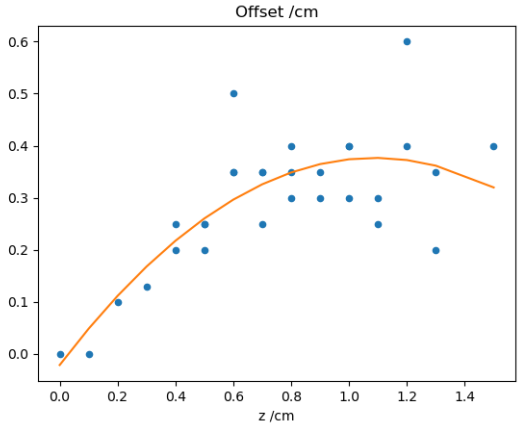


Figure 4.2: The data collected for fitting the model parameters offset for our new virtual brush model (2 outliers were removed).

line and the brush holder after writing finishes for the offset parameter; One example of the many written collections is shown by Figure 4.3 Figure 4.4.

Other parameters such as inertia in Equation 3.10 are hard to estimate in experiments. To decide such parameters, we try different parameters in simulated optimization and choose the ones that end up with the minimum simulation error.

Although we tried a lot for system identification problem, there are also several issues that require further efforts. Although we make good linear assumption on the brush dynamics (especially, the width and drag parameter according to Equation 3.10, Equation 3.11), there are also some observed non-linearity, as shown in Figure 4.5. This figure is extracted from Figure 4.3. By pushing down the brush and lifting it up, we can observe that these two process follows some different linear relationship. This observation also suggests that using a non-parametric model may be better to fit a more accurate brush model.

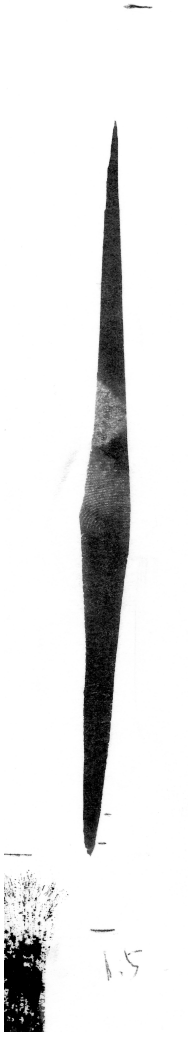


Figure 4.3: One of the written mark pictures used to estimate brush parameters. Although it can be used to estimate both width, drag, and offset, we mainly use it for drag because width and offset are already gotten from the previous method.

Figure 4.4: One of the written mark sets pictures used to estimate brush parameters

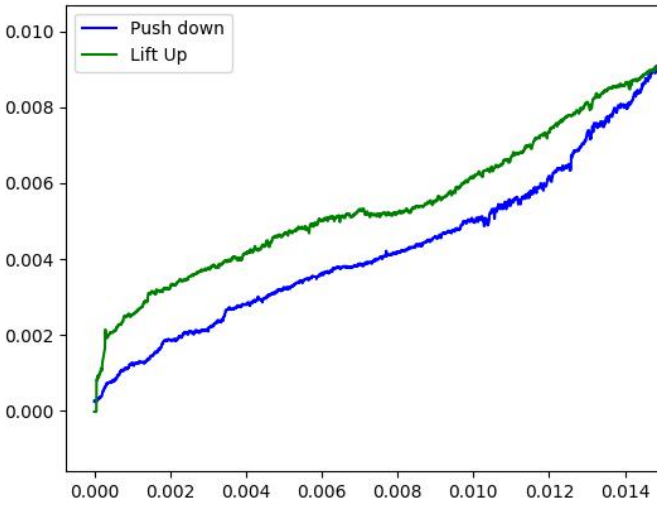


Figure 4.5: Th relationship between width and z under two different writing situations, the figure is gotten from Figure 4.3

CHAPTER 5
TRAJECTORY OPTIMIZATION

Below we formulate the calligraphy writing process as a trajectory optimization problem, and in particular, we adopt some of the machinery from pseudospectral optimal control (PSOC) methods. The chapter is organized as follows: the first section shows a brief review of pseudospectral optimal control, then we present how to use it in the context of calligraphy robots and formulate the trajectory optimization problem. To make the optimization more efficient, we also show how to extract strokes and run optimization based on individual strokes rather than the whole character. Finally, we also introduce our initialization method for the optimization problem and dipping ink method which could improve the Sim2Real gap further.

5.1 Review of pseudospectral optimal control

In this section, we briefly review pseudospectral optimal control methods, closely following the exposition by Fahroo *et al*. [12]. Readers who are already familiar with pseudospectral optimal control can directly jump to the next section.

A simplified version of an optimal control problem can be stated in terms of a cost function C and system dynamics f

$$C = g(x, u) \tag{5.1}$$

$$\dot{x}(t) = f(x(t), u(t), t)), \tag{5.2}$$

where the objective is to find the optimal control sequence $u(t)$ that minimizes the cost function C. Above, $x(t)$ represents the system's state trajectory.

The basic idea in PSOC is to approximate the control trajectory $u(t)$ *and* the state tra-

jectory $x(t)$ by a polynomial curve with unknown parameters, thereby transforming the original problem into a nonlinear programming problem. PSOC method actually belongs to its more general class, spectral methods [37]. Spectral methods are mainly used in finding solutions for partial differential equations, which have similar formulation as the control problem stated above. The basic idea of spectral methods is to represent the control u and system state x by one polynomial curve, and then transform the original problem into a nonlinear programming problem for the best curve parameters. As one special case of spectral methods, the PSOC method specifically chooses the orthogonal polynomials as the basis functions and represents the polynomial curve based on some specifically chosen points [12]. The original curve could be reproduced from these points by Lagrange interpolation easily. Arbitrarily chosen collocation points generally lead to very bad approximation results. The famous "Runge" function is a great example to show this:

$$f(x) = \frac{1}{1 + 25x^2} \tag{5.3}$$

When equally-distanced collocation points are chosen to approximate the Runge function, the approximation curve has a big divergence at the two ends. As such, carefully chosen collocation points are important for the success of the parameterization process.

To this end, pseudospectral methods choose a specific set of points from the curve for interpolation. For example, in the case of Chebyshev pseudospectral methods, the interpolation points used are the Chebyshev-Gauss-Lobatto (CGL) points [38]:

$$t_k = \cos\left(\frac{\pi k}{N}\right), \quad k = 0, ..., N \tag{5.4}$$

To recover the state $x(t)$ at any arbitrary time t one can use barycentric interpolation,

which is a variant of Lagrange interpolation which runs more efficient, i.e.

$$x(t) \approx (\sum_k \frac{w_k}{t - t_k} x_k) / \sum_k \frac{w_k}{t - t_k} \tag{5.5}$$

where

$$w_k = \begin{cases} (-1)^k/2 & k = 0 \text{ or } k = N \\ (-1)^k & \text{otherwise} \end{cases} \tag{5.6}$$

We can now express the original dynamics equation with an approximation, where the objective function (Equation 5.1) can be discretized if necessary. The original problem is thus transformed to minimizing the cost C with respect to the two coefficient vectors $X,\ U$:

$$X = (x_0, ..., x_N),\ U = (u_0, ..., u_N) \tag{5.7}$$

representing the values of the states and controls, respectively, at the CGL points.

5.2 PSOC for calligraphy

Below we apply these methods to trajectory optimization for open-loop control trajectories of a robot end-effector, with the goal of faithfully reproducing Chinese characters. PSOC methods are generally used for collocated optimal control where the system dynamics are enforced through specialized components of the cost function. However, because our control input is simply the trajectory velocity, which can be calculated exactly from the end-effector trajectory, we have no need for separate control parameters U.

The optimization for a character is decomposed into a series of trajectory optimization problems corresponding to individual strokes of the character. This greatly simplifies the process and is also more computationally efficient. In this, we are helped by the existence of vector-based character databases in which characters are stored as their decomposed

individual strokes, as described next.

5.3 Stroke extraction

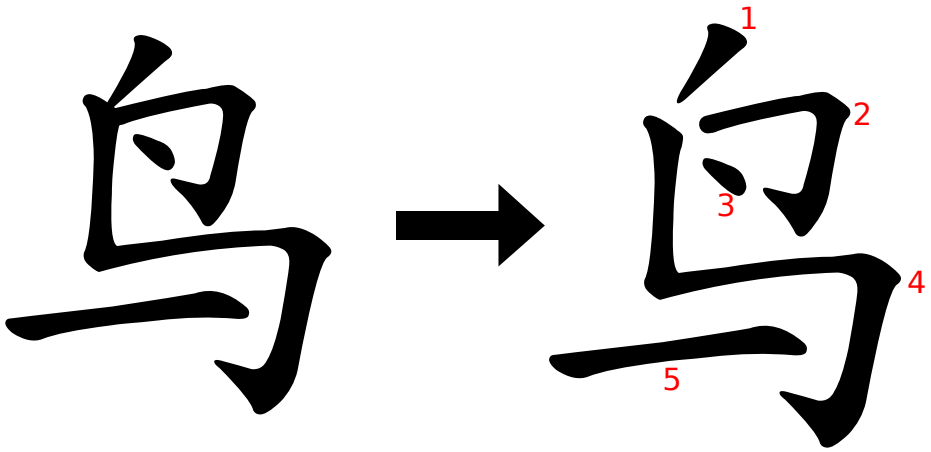


Figure 5.1: The character ‘bird’, meaning ‘niao’, and its extracted strokes

To obtain reference images for each stroke, as well as initialize the nonlinear optimization, stroke extraction is a necessary step for the whole calligraphy project. There is much research devoted to finding an efficient and effective algorithm for stroke extraction, which we already present a complete review of the related work section.

In our case, we exploit the existence of vector-based character databases. Vector-based images (also known as vector graphics) are a special type of computer graphics images that are defined by geometric primitives, such as points, lines, and simple curves (mostly, Bezier curves). Such basic components are well defined and easily parameterized by a set of control points, and can also be combined to form various polygons and almost all kinds of shapes with an approximation to a degree. Vector graphics are also well known for their unique advantage to be scaled up or down without any blurry issues. They are widely adopted in many applications in the form of “SVG”, “EPS”, “PDF”, etc.

Vector-based images are advantageous because the Chinese characters are usually stored by strokes. By generating Bezier curves around the profile of strokes, many beautiful artistic styles of Chinese calligraphy can be explored. Such design convenience makes the

stroke extraction a trivial task. In this thesis, the database we choose is a Scalable Vector Graphics (SVG) database from MakeMeHanzi [39], and an example of extracted strokes can be seen in Figure 5.1.

5.4 Stroke Trajectory Representation

The stroke trajectories will be used as the open-loop control trajectories for the robot to draw strokes, and we represent them as three-dimensional trajectories of the end-effector. Each of the x, y, z components are separately represented as 1-dimensional Chebyshev polynomial curves. Chebyshev polynomials are continuous, differentiable curves, and thus pretty suitable to represent stroke trajectories and also for optimization. The trajectories $\boldsymbol{X}$ with three dimensions x, y, z are expanded by interpolating the values at the Chebyshev-Gauss-Lobatto (CGL) points given by Eq. Equation 5.5 and Eq. Equation 5.6:

$$\boldsymbol{X}(t) \approx (\sum_{k} \frac{w_k}{t - t_k} \boldsymbol{X}_k) / \sum_{k} \frac{w_k}{t - t_k} \tag{5.8}$$

Hence, the decision variables are the combination of the three sets of CGL points in the x, y, z dimensions:

$$X = (x_0, ..., x_N; y_0, ..., y_N; z_0, ..., z_N) \tag{5.9}$$

5.5 Stroke Trajectory Simulation

In Chapter 3, we introduce the virtual brush model which is able to simulate the writing behavior of the real brushes. The proposed dynamic model updates the parameters of the virtual brush at each time step following:

$$\mathcal{X}_{i+1} = f(\mathcal{X}_i, \mathcal{U}_i) \tag{5.10}$$

where the specific form of the nonlinear function $f(\mathcal{X}_i, \mathcal{U}_i)$ is given by at the end of chapter 3. However, this form of the virtual brush is not very useful until it could be used by the optimization system. To be more specific, given a stroke parameterized by stroke parameters X_s, we would like to obtain the simulation picture corresponding to the stroke. The function that the optimization system is more interested with is the one as follows:

$$V(X_s) = I \tag{5.11}$$

where $V(X_s)$ represents the final virtual brush model, I represents the simulated picture in matrix form. To establish the connection from the nonlinear function f in Equation 3.5 to the Equation 5.11, we decide to run discrete simulation. The stroke trajectory X_s is sampled evenly by a specific number M of control points x_i, y_i, z_i. By assuming the brush always executes at a constant speed, the proposed virtual brush model is able to render the written mark for each of the control points one by one, and finally, we can get the simulation picture by overlapping the sampled control points:

$$V(X_s) = \sum_{i}^{M} D(\mathcal{X}_i) \tag{5.12}$$

where i denotes each time instance, and the drawing function D is gotten from the Drawing component section in Chapter 3, $\mathcal{X}_i$ is decided from Equation 3.5 iteratively.

5.6 Stroke Trajectory Optimization

We optimize trajectories by using the virtual brush and initialized trajectory to create a simulated image, which we then compare with the ideal image and try to adjust the trajectory to minimize the differences. The comparisons between simulation and ideal are made at a stroke level which both improves the speed and performance of our optimization. To be more specific, the objective function C_s for stroke s minimizes the sum-squared pixel difference between the image $V(X_s)$ produced by simulating the drawing process, and a

reference image I_s created from the SVG representation. The optimal trajectory parameters X_s^* for stroke s are obtained as

$$X_s^* = \arg\min_{X_s} C_s(X_s) \tag{5.13}$$

$$= \arg\min_{X_s} \{\| V(X_s) - I_s\|^2 + \sum_k \beta_k \|z_k\|^2\} \tag{5.14}$$

Above, $V(\cdot)$ is a function that represents the virtual brush simulation, taking a stroke trajectory with pseudospectral parameterization X_s and drawing a stroke image according to the given trajectory. The $\beta_k\|z_k\|^2$ are regularization terms on the brush height to make sure the trajectory has a tendency to lift up the brush near the end of the stroke. The weights β_k are determined by experiments.

Actually, in the original calligraphy problem, there are a few implicit geometric constraints, like the brush model should not penetrate below the paper surface. We didn't explicitly include these constraints into our optimization problem because they are generally never violated. For example, it requires the brush to dip very low so that it could penetrate the paper, which, however, will generate very bad simulation results with very big simulation error because of the virtual brush model. In other words, our virtual brush model could serve similarly as a barrier function in classical convex optimization problems to satisfy implicit constraints.

Generally, there are many choices in deciding optimization parameters when a nonlinear optimization program is formulated. In practice, we try all polynomial orders between 3 and 8 for each stroke and pick the one with the smallest error, in order to adapt to strokes of varying length and complexity. A dense and even sampling of the polynomial is performed in V to execute a continuous stroke in simulation.

5.6.1 Nonlinear optimization review

The optimization objective function Equation 5.14 is a standard nonlinear optimization problem, and can be solved efficiently via gradient-based methods. There are many popular gradient-based optimization algorithms, and we will introduce them one by one.

Problem description and approximation

We assume the nonlinear optimization problem of interests as an unconstrained least-square optimization problem:

$$\min_x r(x)^T r(x) \tag{5.15}$$

where

$$r(x) = f(x) - b \tag{5.16}$$

is a continuous and differentiable function, and x could be a multi-dimensional variable. We want to find an optimal x to achieve the minimum possible value of $f(x)$. In the calligraphy problem, Equation 5.15 corresponds to Equation 5.14. Also note that the regularization term can be easily included as new dimension of the nonlinear function $f(x)$ to be consistent.

For notational convenience, we denote the first-order derivative matrix (i.e., the Jacobian matrix) and the second order derivative matrix (i.e., the Hessian matrix) as follows:

$$f'(x_0) = \mathbf{J} \tag{5.17}$$

$$f(x_0)'' = \mathbf{H} \tag{5.18}$$

The Jacobian matrix $\mathbf{J}$ is computed using numerical differentiation at each iteration, and the Hessian matrix is approximated as $\mathbf{J}^T\mathbf{J}$.

Gradient descent

The nonlinear function Equation 5.15 can be locally approximated by first order derivatives based on Taylor expansion. If we only take the first-order derivatives, then we have

$$f(x) \approx f(x_0) + \mathbf{J}(x - x_0) \tag{5.19}$$

As such, beginning with x_0, if we want to find the optimal x to minimize the objective $f(x)$, we should go along the inverse of the derivative:

$$x = x_0 - \gamma \mathbf{J}^T b \tag{5.20}$$

where $gamma$ decides how long a step do we want to take. If we move too far away, then our original first-order approximation loses its effect, and so we cannot promise the inverse gradient still provides the best decision anymore.

After finishing the update for one step, we can still adopt the similar method and use the first-order derivative to locally approximate the new position, and so we can iteratively end up with a value x^* where we cannot or should not move forward (like, the first-order derivative $f`(x)$ is pretty small), and that would be our optimal estimate for the original nonlinear optimization problem.

Gaussian-Newton method

The nonlinear function Equation 5.15 can also be locally approximated by second order derivatives based on Taylor expansion:

$$f(x) \approx f(x_0) + \mathbf{J}(x - x_0) + \frac{1}{2}(x - x_0)^T \mathbf{H}(x - x_0) \tag{5.21}$$

If we decide to adopt the second-order derivative, usually, the step size to move is fixed because the optimal solution of a quadratic function can be easily gotten, i.e.,

$$x = x_0 - (\mathbf{J}^T\mathbf{J}J)^{-1}\mathbf{J}^T b \tag{5.22}$$

Levenberg-Marquardt (LM) algorithm

Levenberg-Marquardt (LM) algorithm is a second-order trust-region method. In short, LM switches between a gradient-based search and a second-order Gauss-Newton update by controlling a *damping factor* λ:

$$[\mathbf{J}^T\mathbf{J} + \lambda \mathbf{diag}(\mathbf{J}^T\mathbf{J})]\delta = \mathbf{J}^T[V(c_i) - I_i] \tag{5.23}$$

$$x = x_0 - \delta \tag{5.24}$$

The intuition behind Levenberg-Marquardt algorithm is that the gradients near the optimal value (which ideally has 0 gradients) are usually small because the objective function is continuous and differentiable. As such, gradient descent in such case usually takes many steps to reach the optimal value, as such Newton optimizer is preferred because it is able to reach the optimal value in one step if only the quadratic approximation holds well (and it does have a high probability to hold because the first-order derivative is very small in the adjacent region of optimal value).

However, when the iteration point is far from the optimal value, gradient descent usually could take a longer step, and so faster convergence than the Newton method, and so is preferred. The damping factor λ is simply adjusted based on this assumption. At the beginning of optimization, it is usually set as a large value, and so the LM algorithm behaves more like a gradient descent algorithm; and then after each successful iteration, $lambda$ decreases for one order, and so the whole LM algorithm gradually behaves more like New-

ton's method.

In experiments, we tried both algorithms, and the LM algorithm performs substantially better than the gradient descent and Newton's method. We use the GTSAM library [40] to perform the optimization. GTSAM was originally created to solve simultaneous localization and mapping problems but has been used in many different contexts since, including motion planning [41, 42].

5.7 Stoke Trajectory Initialization

We use the skeleton of a stroke to initialize the x and y coordinates, while the initial z coordinates are set to a fixed value. From our observations, when people write calligraphy, they generally make the brush approximately follow the skeleton of the stroke while varying the height of the brush.

Because we start from a vector-based representation, extracting an initial trajectory for the individual strokes is much simplified. Even when starting from images, there are many good image-based skeleton extraction algorithms, e.g. the Chordal Axis Transform [43, 44], an example result of which is shown in Figure 5.2a. In our case, we use the "animation path" provided by the database as the initial estimation for the 2D x_i and y_i sequences for simplicity. CGL points are sampled on each skeleton path to obtain the pseudospectral representation. Generating an easy estimate for z is not intuitive, and so we just initialize it with a constant sequence.

Given the initial trajectory, we can simulate the image formation process using both the simple virtual brush and the dynamic virtual brush, as illustrated in Figure 5.2b and Figure 5.2c respectively. The position of the written mark from the dynamic virtual brush is different from its given trajectory, which is ignored by most previous research.

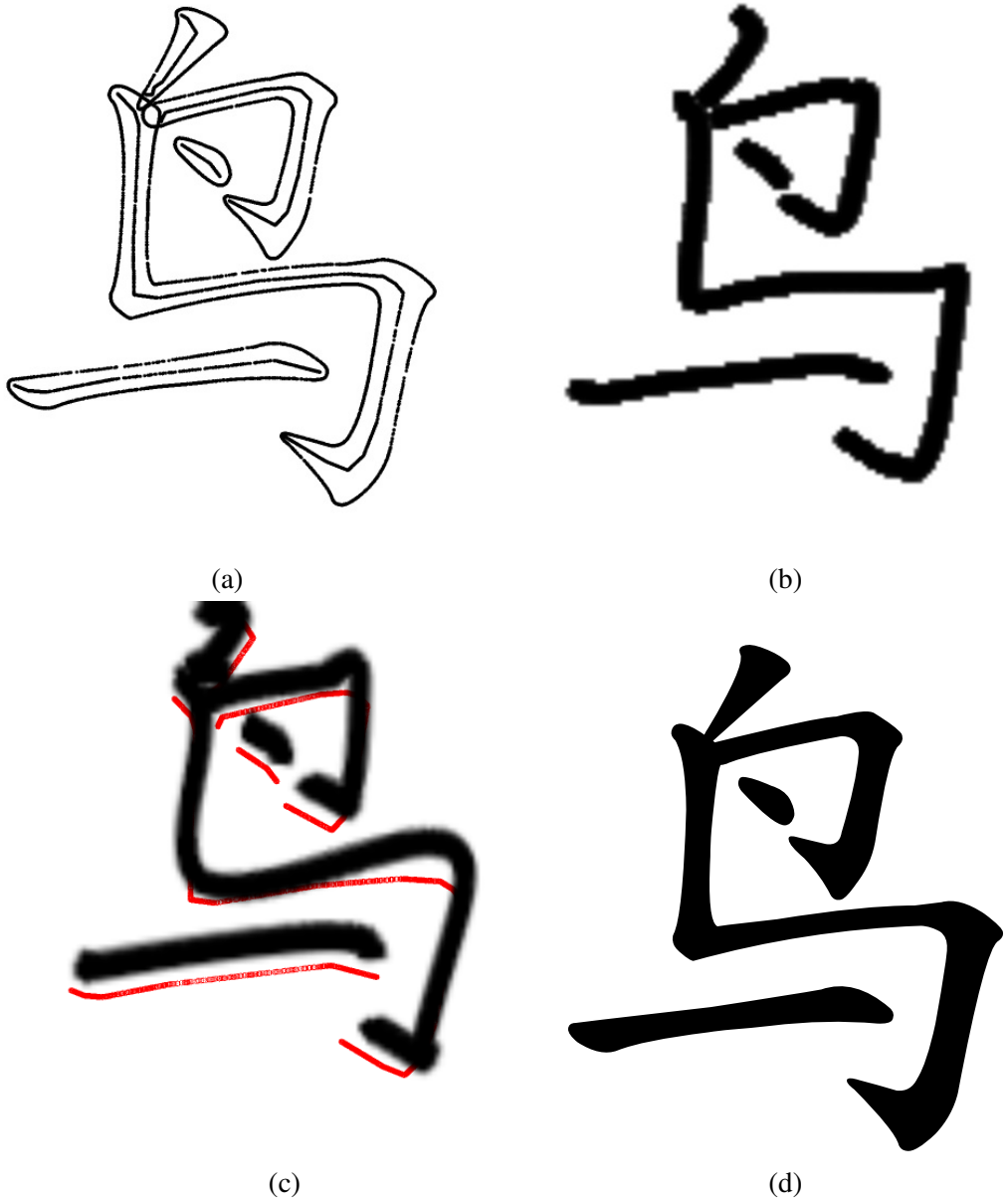


Figure 5.2: (a) The character ‘bird’ with its skeleton; (b) Initial image from the simple virtual brush; (c) Initial image from the dynamic virtual brush with the trajectory (red) (d) The reference picture

CHAPTER 6

RESULTS

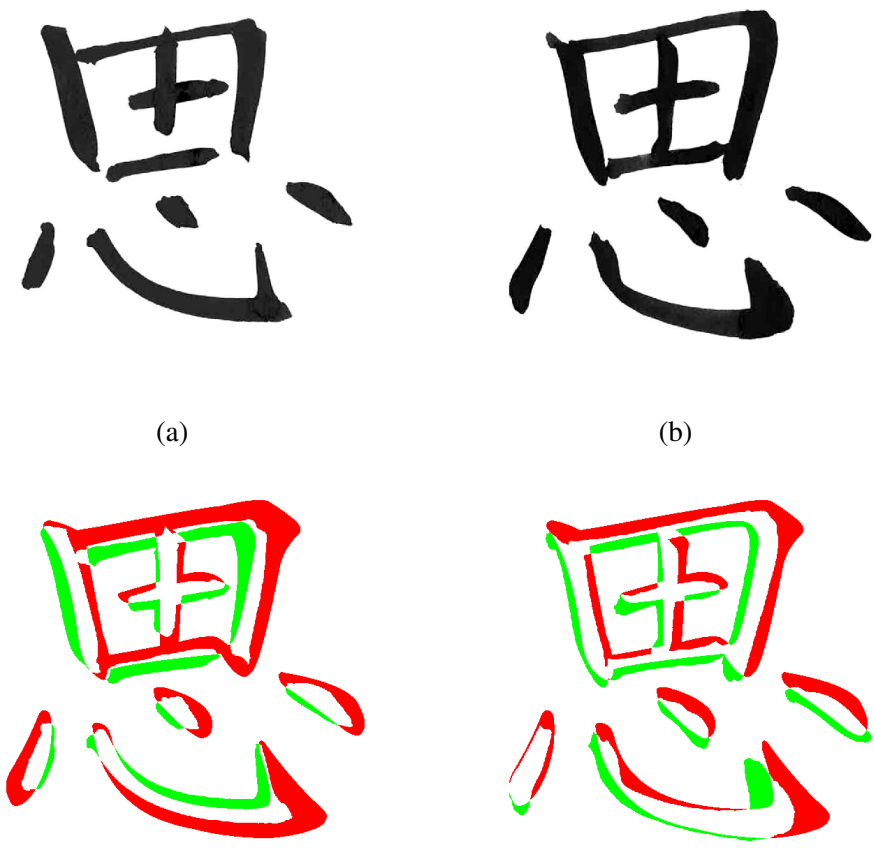
Below we present the results of our approach, including photographs of characters that have been drawn by a Franka robot, using inverse kinematics provided by MoveIt![45]. The characters are written using relatively slow end-effector velocity to prevent excessive jerk and vibrations. Some of the results (Figure 6.1, Figure 6.3, Figure 6.2) in this chapter are already published in the main author's IROS paper [13].

6.1 Comparison between simple and dynamic virtual brush model

Figure 6.1 shows a comparison between the written results generated by the simple virtual brush model and by the dynamic virtual brush model. Both simulation images generated by the two virtual brush models look very good and so are not added there. The simple virtual brush model is able to generate good touching and brush outlines for some simple strokes (like a dot) because of the good linear fitting model in the Sim2Real section, but not all (like the long hook). However, it can be seen that the dynamic virtual brush shows obvious advantages in terms of preserving the relative positions of strokes since it models the deformation of the real brush. Periodically restoring the brush to its initial state helps the simple brush to behave better, but it is still worse than the dynamic virtual brush model.

6.2 Comparison before and after optimization for characters

In Figure 6.3, we show more extensive results for three different characters. Both the simulated and written images before and after optimization are shown for easy comparison. From the figure, we can see that the optimization achieves good performance for simulated images. However, because of a Sim2Real gap in the virtual brush model, the proposed



(a) (b)

(c) (d)

Figure 6.1: Character ‘si’, meaning ‘think’. (a) Written result following *simple* brush optimization; (b) Written result following *dynamic* brush optimization; (c) Written result from (a) overlaid with original SVG file; (d) Written result from (b) overlaid with original SVG file. The green and red pixels represent positive and negative differences respectively.

method still has a way to go in terms of approaching the smoothness and definition of detail displayed in the reference images, rendered from the vector-based character database.

A detailed error analysis shows that the optimization achieves almost perfect results in simulation, but that there still is a considerable Sim2Real gap. Figure 6.2 shows the differences between reference images and both simulated and written characters, respectively before and after optimization. As in Figure Figure 6.1, green and red pixels represent positive and negative differences. Column (c) shows that the optimization can achieve near-perfect results, where the only errors remaining stem from anti-aliasing, as well as a few

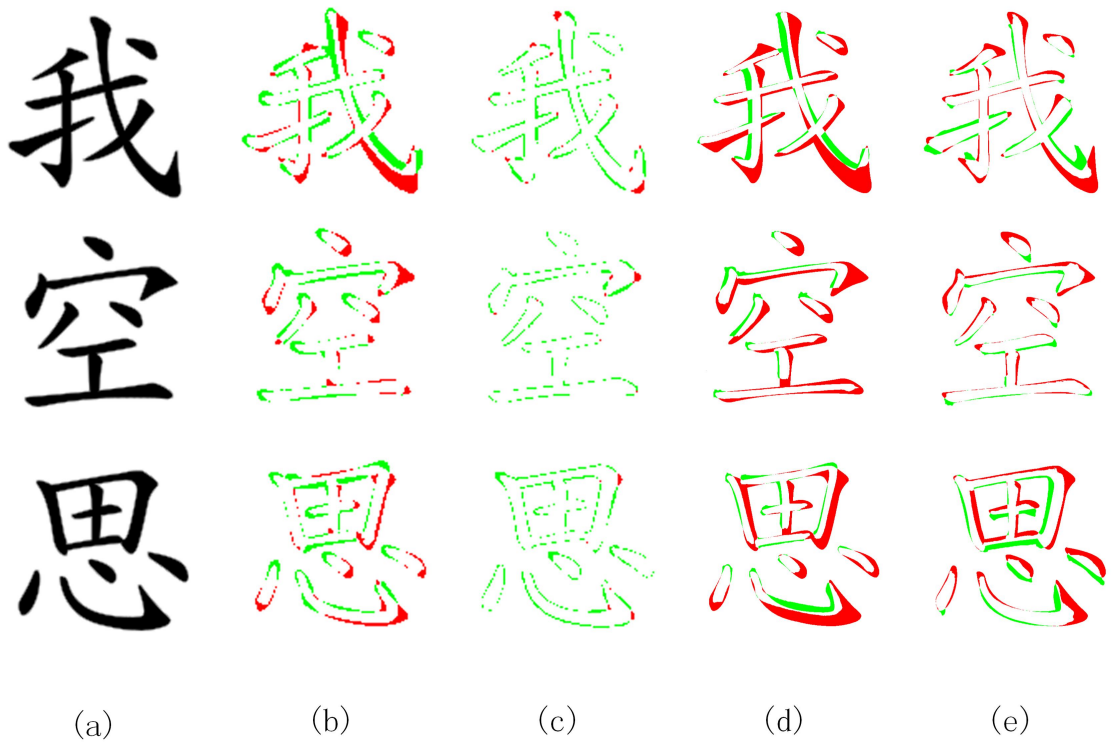


Figure 6.2: Error analysis for the three characters from Figure Figure 6.3. As before, green and red pixels represent positive and negative differences, respectively. (a) The original character pictures from the database; (b) Simulated image drawn by the dynamic virtual brush using the initial trajectory; (c) Simulated image drawn by the dynamic virtual brush using an *optimized* trajectory; (d) Actual written image following initial trajectories; (e) Actual written image following the *optimized* trajectories.

small errors near the stroke endings. However, we see larger errors when these optimized results are executed on the robot (column e), even though the characters themselves look visually pleasing.

Our system achieves much faster speed compared to previous work. For example, Kwok *et al.* [8] reports that the average running time for optimization of one character is 15-25 minutes, while our method optimizes a complete character in about 25 seconds on average. That shows an absolute advantage even considering the hardware and software influence (e.g. slightly better CPU, Matlab vs. C++ code).

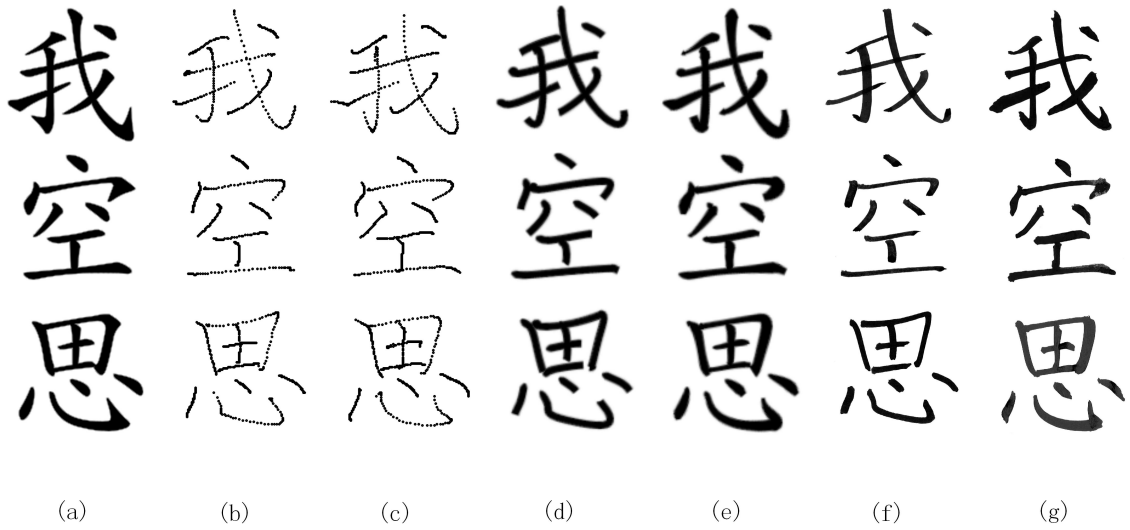


Figure 6.3: The optimization of different characters: from top to down, ‘wo’, ‘kong’, and ‘si’, meaning ‘me’, ‘empty’, and ‘think’. (a) The original character pictures from the database; (b) Initial trajectory estimates; (c) The trajectory obtained from optimization; (d) Simulated image drawn by the virtual brush using the initial trajectory; (e) Simulated image drawn by the virtual brush using an *optimized* trajectory; (f) Written image following initial trajectories; (g) Written image following the *optimized* trajectories.

6.3 More simulation results and discussion

These figures about ‘one stone two birds‘ show more results about the optimization process. We can see that easy characters such as ‘one‘ could generate extremely good simulation results, while some complicated character like ‘bird‘ has some worse results. However, the failure of ‘bird‘ should be mostly caused by the limitation of our current implementation rather than the proposed algorithm in this thesis. Currently, we use the same order of Chebyshev polynomials for all the strokes in one single character. As such, characters composed of both simple and complicated strokes usually have to adopt a high order of Chebyshev polynomials for optimization otherwise the complicated strokes would have very bad simulation results. However, it adds a lot of difficulty for simple strokes partly because the Jacobian matrix during optimization becomes less sensitive to changes on Chebyshev polynomials. In other words, the Jacobian matrix becomes smaller as unnecessary orders are added. As such, a better way to deal with this situation would be adopting different

(a) (b)

(c) (d)

Figure 6.4: Characters ‘yi, shi, er, niao’, meaning ‘one stone two birds’. These characters are generated by the initial estimation from simulation

orders of Chebyshev polynomials for different strokes. Unfortunately, we don’t have time to implement that before writing this thesis.

6.4 Improving the Sim2Real Gap

6.4.1 Dipping ink

Part of the Sim2Real gap is due to the unpredictable state of the brush after executing a stroke. After the brush writes a complicated stroke the tip may become messy. At that

(a) (b)

(c) (d)

Figure 6.5: Characters ‘yi, shi, er, niao’, meaning ‘one stone two birds’. These characters are generated after optimization from simulation

time, accurately predicting the state of the brush and finding a feasible control trajectory is difficult and unreliable. To avoid accumulating prediction error as more strokes are written, we have the robot *dip ink* after a stroke is written, which restores the brush to a predictable state. We handcrafted a control algorithm to accomplish this: given a circular inkstone, the brush is pushed down heavily at first to make the tip flat, and we then slowly move it to the edge of the inkstone in different directions with a gradually smaller extent. After that, the tip is generally restored to a predictable state, which we reflect in the initial state $\mathcal{X}_0$ of the

dynamic brush model. In particular, width w, drag d, offset o, are set to zero, whereas the orientation θ is initialized to align with the initial trajectory direction.

To highlight the influence of dipping ink, these two figures Figure 6.6 show the second dot of writing the character 'si'. After the complicated stroke before the second dot stroke, the tip becomes messy, and dipping ink at this time could help to get a better dot. Although dipping ink won't bring improvements to all the strokes, some strokes could get obvious improvement.

(a) (b) (c)

Figure 6.6: (a) The reference dot from character 'si'. (b) written dot without dipping ink (c) written dot with dip ink. We can see the obvious improvement posed by dipping ink before writing this stroke.

6.5 Fetch robots setup and discussion

The first robot that we work with is the fetch robot [46]. However, after some careful experiments, we come to the conclusion that the Fetch robot is not suitable for the calligraphy task, and so turn to work with our current Panda robot. In this section, I describe the procedure to use the Fetch robot to do the calligraphy work. The setup picture Figure 6.7 is also added. Almost all the procedures in the programming part are well explained in the official calligraphy repository's README.md, and so I only add the manipulation procedure to use Fetch robot there:

- Open the power for Fetch robot, connect a monitor, keyboard, and mouse to it

Figure 6.7: Fetch writing configuration

- Choose to use local control or remote control via ssh
- Make sure MoveIt is well installed
- Open MoveIt environment
- Run the python script for the robot to execute trajectories, which can be found in the calligraphy repository

The main problem with the Fetch robot comes from its unstable base. Thanks to Gerry's

Figure 6.8: Fetch written results, “bird” in Chinese

great observation, we find that the base shakes a lot during the writing process. Although the robot arm is very stable, shaking base make the arm shake too, and so become not suitable for the highly accurate and stable task, such as the calligraphy work. This becomes the main reason for the failure of our previously submitted paper because the difference before and after optimization is not obvious. However, things become much better after we switch to a new, stable robot: Panda robot!

6.6 Robots setup - Panda

The main problem with the Fetch robot comes from its unstable base. Thanks to Gerry’s great observation, we find that the base shakes a lot during the writing process. Although the robot arm is very stable, the shaking base makes the arm shake too, and so becomes not

Figure 6.9: Fetch written results, “empty” in Chinese

suitable for the highly accurate and stable task, such as calligraphy work. This becomes the main reason for the failure of our previously submitted paper because the difference before and after optimization is not obvious. However, things become much better after we switch to a new, stable robot: Panda robot!

- Open the power for Panda robot, make sure a desktop is setup and connected to it
- Decide control methods, you can use Panda’s control library or MoveIt for motion planning
- Make sure MoveIt is well installed
- Open MoveIt environment

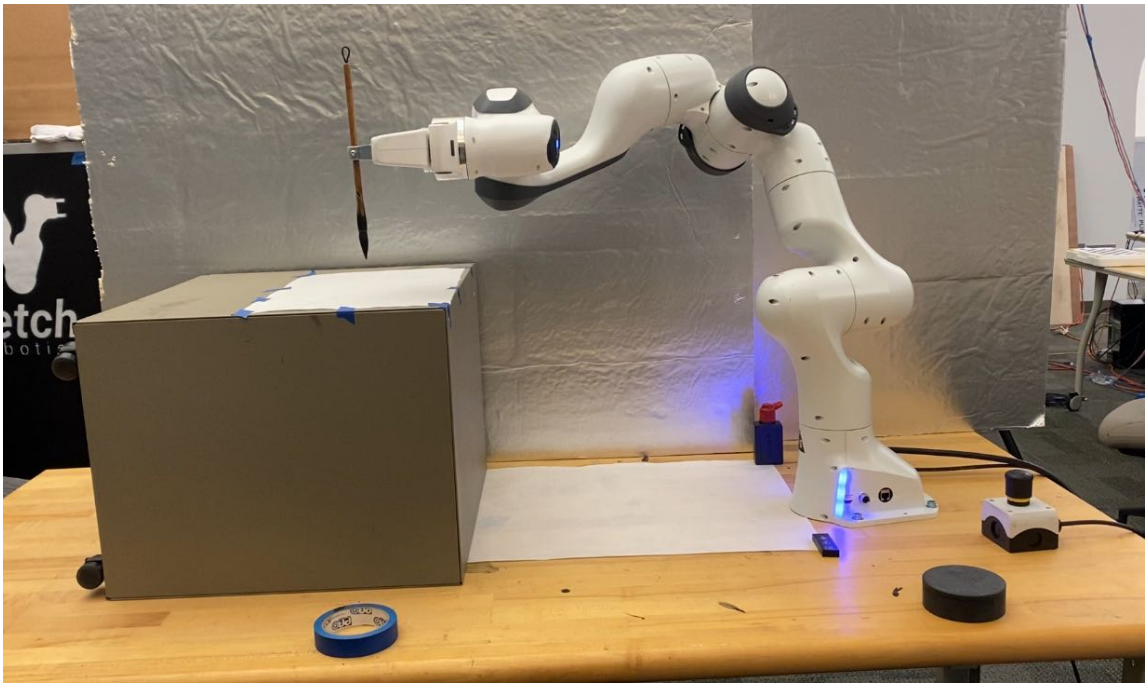


Figure 6.10: Panda writing configuration

- Run the python script for the robot to execute trajectories, which can be found in the calligraphy repository

Finally, all the results generated by these two robots are either at the lab or have been thrown away before I write this thesis.

CHAPTER 7
FAILURE EXPERIENCE

In this chapter, I discuss something that I tried but did not succeed before.

7.1 Chu's virtual brush model

Chu *et al*. propose an interesting virtual brush model for digital painting at around 2004 [10]. The virtual brush model is built based on an idea similar to finite element method, though the system dynamics are predicted via the energy minimization principle. Although Chu's model is much more complicated than our proposed model in the previous chapter, it provides a more detailed description of both the dynamics process and the possible written mark. As such, I also try to implement a *simplified* version of that virtual brush model around January 2020. However, the trajectory optimization process is extremely slow because of the complicated dynamics concerned, and the final result is also not as good as our previous brush model. I believe this failure experience also proves the importance of an efficient virtual brush model to calligraphy robots. In the next several sections, I would show some more rendered results from me.

7.1.1 Initial estimate based on Chu's model

Figure 7.1 shows the initial simulation pictures generated by my implementation of Chu's brush model. These simulation results are rendered via the initial trajectory extracted. Actually, these results are good enough to serve as initial pictures, although usually, they take a long time to run.

7.1.2 Optimization based on Chu's model

Figure 7.2 and Figure 7.3 show some optimization results generated by Chu's model.

Figure 7.1: Figure (a) shows the reference picture for character ‘zhi‘, figure (b) and (c) show the initial simulation pictures generated by the Chu model with different sets of brush parameters.

Figure 7.2: Simulation for a turn based on Chu’s model. The gray trajectories are the control input, and the black trajectories are the simulated trajectories. The three figures use different set of optimization parameters.

Figure 7.3: Simulating the character ’one’ by Chu’s model. The left one is the reference pictures, the right two are the simulation results generated via different optimization parameters.

7.1.3 Analysis based on Chu's model

From the results shown in the previous sections, it can be easily seen that my implementation on Chu's model does not work well. There are several possible reasons for that, which I summarize as follows:

- Programming issues. My code may have bugs, and may not be written in the most efficient way. These things would partly be the reasons for the bad results.
- Model complexity. Chu's model itself is involved with a complicated mechanism and has a much higher calculation cost than the dynamic model introduced in chapter 3. Its most writing behavior follows a nonlinear relationship, which also adds difficulty to the optimization process.
- Paper deficiency. In the original paper written by Chu, many important details are ignored such as the specific form for strain and friction energy functions. As such, during my implementation, I can only try to find a reasonable way to formulate these energy functions, which may also be the cause of the bad final performance.

CHAPTER 8

CONCLUSION

In this paper, we present a trajectory optimization method to control robots to write calligraphy. During the optimization process, we search for open-loop control trajectories which the virtual brush could use to generate a simulation picture as similar as the original picture. Pseudospectral methods are adopted from the optimal control literature to parameterize strokes to achieve very efficient and fast optimization. Apart from it, we also propose a novel virtual brush model to simulate the behavior of real brush models. The virtual brush model plays an important role in guiding the optimization process, and so its accuracy mostly decides the performance of final results. Our experiments show that the proposed dynamic virtual brush model yields good results in simulation and in turn produces reasonable open-loop control trajectories for a real robot.

CHAPTER 9
FUTURE WORK

In this thesis, we presented a trajectory optimization method to make robots write calligraphy, searching for open-loop control trajectories to approximate a reference image. The proposed dynamic virtual brush model yields good results in simulation and in turn, produces reasonable open-loop control trajectories for a real robot. However, from the results, it is clear that the Sim2Real gap has not been fully closed, and it is an open question of whether a closed-loop strategy will ever yield master-level calligraphy. As such, in future work, we plan to explore the following aspects.

9.1 Building more accurate brush model

The central component that influences the Sim2Real gap in the calligraphy project is the brush model. Having an accurate brush model has two requirements. At first, the proposed brush model should be able to accurately predict system dynamics given proper initial conditions and control input. Secondly, the proposed brush model parameters should be able to fit into a specific real writing brush so that it is able to work with this specific writing brush. The first question can be summarized as dynamics simulation, and the second can be summarized as system identification.

To solve both problems and close the Sim2Real gap, we plan to adopt port-Hamiltonian framework to model the brush systems in the near future. The key factor to decide brush dynamics is Hamiltonian, which can be learned from real writing data, or to be more specific, writing pictures. As such, we can integrate the two questions into the same framework and solve them efficiently. This port-Hamiltonian system would be a complex system with many components interacting with each other, and so using factor graph would be a natural choice for both description and efficient solution. Similar as Chu model as we described in

the previous chapter, we can also use the finite element method to model the brush, and get system dynamics via inference easily.

9.2 Speeding up the optimization process

The optimization part, from the main author's previous experience, would be pretty slow if no specific changes are introduced. Because of the existence of picture difference error, the whole optimization is involved with solving the inverse of a large and dense Hessian matrix. However, this big dense Hessian matrix is actually the addition of a sparse matrix and a low-rank matrix, which means great speed-up could be exploited.

9.2.1 Matrix inverse lemma

We adopt the matrix inverse lemma from Stephen Boyd's Convex Optimization [47], and illustrate it as follows:

$$(A + BC)^{-1} = A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1} \tag{9.1}$$

where $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times p}$, $C \in \mathbf{R}^{p\times n}$.

In other words, if $p \ll n$, we can easily get $(A + BC)^{-1}$ from A^{-1} with trivial cost; Also, the calculation cost above without considering finding A^{-1} is $O(pn^2)$ if we ignore the sparse part of A.

9.2.2 Revisit the standard least square problem

Let's revisit our objective following the formulation used in [48]:

$$\min_{\Delta} \|A\Delta - b\|_2^2 \tag{9.2}$$

where $A \in \mathbf{R}^{m \times n}$. And one of the common way to solve for it via pseudo-inverse matrix:

$$\Delta = (A^T A)^{-1} A^T b \tag{9.3}$$

To solve for the big inverse matrix, we can use Cholesky decomposition to get $A^T A = R^T R$, where R is a triangle matrix of size $n \times n$;

In reality, the calculation above can speed up a lot if the matrix A shows some sparse structure.

9.2.3 Application in least square and the calligraphy problem

We consider the application in some special cases, where only a few global constraints are added to a sparse situation. It corresponding to adding a few global factors in factor graph. That means, we can decompose the A matrix as follows:

$$A = \begin{bmatrix} A_0 \\ u \end{bmatrix} \tag{9.4}$$

where $A_0 \in \mathbf{R}^{m \times n}$ is a sparse matrix , and $u \in \mathbf{R}^{p \times n}$ is a fat matrix with $p \ll n$; In that case, we have

$$(A^T A)^{-1} = (\begin{bmatrix} A_0 \\ u \end{bmatrix}^T \begin{bmatrix} A_0 \\ u \end{bmatrix})^{-1} = (A_0^T A_0 + u^T u)^{-1} \tag{9.5}$$

where the lemma can be applied perfectly. As such, we can expect to find the inverse of $A^T A$ approximately at the cost of finding the inverse of $A_0^T A_0$, where A_0 is be a sparse matrix. The added cost would be linearly proportional to the dimension of u, i.e., p. To solve the original least square problem, the cost would be $O(A_0) + O(mn^2) + O(pn^2)$ as opposed to $O(n^3)$ in the original setting.

And this is exactly the same situation that we just mention for the calligraphy situation.

The picture difference error during the writing process corresponds to the global constraints or the B and C matrix in the matrix inverse lemma, and so we can exploit this lemma to solve for the whole optimization process much faster.

To summarize, in future work we plan to investigate both more accurate brush models, better system identification techniques, and improving calculation cost. Finally, we may also consider sensor fusion, such as force or position detection during to improve the system performance even further.

REFERENCES

[1] W. Li, "Chinese writing and calligraphy," in *Chinese Writing and Calligraphy*, 2010.

[2] Y. Sun and Y. Xu, "A calligraphy robot — callibot: Design, analysis and applications," *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 185–190, 2013.

[3] Y. Sun, H. Qian, and Y. Xu, "Robot learns Chinese calligraphy from demonstrations," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2014, pp. 4408–4413.

[4] A. Kotani and S. Tellex, "Teaching robots to draw," *2019 Intl. Conf. on Robotics and Automation (ICRA)*, pp. 4797–4803, 2019.

[5] S. Mueller, N. Huebel, M. Waibel, and R. D'Andrea, "Robotic calligraphy - learning how to write single strokes of Chinese and Japanese characters," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2013, pp. 1734–1739.

[6] K.-W. Kwok, S. M. Wong, K. W. Lo, and Y. Yam, "Genetic algorithm-based brush stroke generation for replication of chinese calligraphic character," *2006 IEEE International Conference on Evolutionary Computation*, pp. 1057–1064, 2006.

[7] J. H. M. Lam and Y. Yam, "Stroke trajectory generation experiment for a robotic chinese calligrapher using a geometric brush footprint model," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2315–2320, 2009.

[8] K. W. Kwok, K. W. Lo, S. M. Wong, and Y. Yam, "Evolutionary replication of calligraphic characters by a robot drawing platform using experimentally acquired brush footprint," in *2006 IEEE Intl. Conf. on Automation Science and Engineering*, IEEE, 2006, pp. 466–471.

[9] S. Xu, F. C. M. Lau, and Y. Pan, *A computational approach to digital Chinese painting and calligraphy*. Springer Science & Business Media, 2009.

[10] N. S.-H. Chu and C.-L. Tai, "Real-time painting with an expressive virtual chinese brush," *IEEE Computer Graphics and Applications*, vol. 24, pp. 76–85, 2004.

[11] G. N. Elnagar and M. A. Kazemi, "Pseudospectral chebyshev optimal control of constrained nonlinear dynamical systems," *Computational Optimization and Applications*, vol. 11, pp. 195–217, 1998.

[12] F. Fahroo and I. M. Ross, "Direct trajectory optimization by a chebyshev pseudospectral method," *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, vol. 6, 3860–3864 vol.6, 2000.

[13] S. Wang, J. Chen, X. Deng, S. Hutchinson, and F. Dellaert, "Robot calligraphy using pseudospectral optimal control in conjunction with a novel dynamic brush model," *ArXiv*, vol. abs/2003.01565, 2020.

[14] S. Kudoh, K. Ogawara, M. Ruchanurucks, and K. Ikeuchi, "Painter robot: Manipulation of paintbrush by force and visual feedback," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS) Workshop" Art and Robots*, 2007, pp. 63–68.

[15] Y. Lu, J. H. M. Lam, and Y. Yam, "Preliminary study on vision-based pen-and-ink drawing by a robotic manipulator," *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 578–583, 2009.

[16] L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina, "Watercolour robotic painting: A novel automatic system for artistic rendering," *Journal of Intelligent and Robotic Systems*, pp. 1–16, 2018.

[17] X. Niu, J. Liu, L. Sun, Z. Liu, and X. Chen, "Robot 3d sculpturing based on extracted nurbs," *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1936–1941, 2007.

[18] Y. Jun, G. Jang, B.-K. Cho, J. Trubatch, I. Kim, S.-D. Seo, and P. Y. Oh, "A humanoid doing an artistic work - graffiti on the wall," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1538–1543, 2016.

[19] Y. Sun and Y. Xu, "A calligraphy robot Callibot: Design, analysis and applications," in *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, IEEE, 2013, pp. 185–190.

[20] K. Sasaki, K. Noda, and T. Ogata, "Visual motor integration of robot's drawing behavior using recurrent neural network," *Robotics and Autonomous Systems*, vol. 86, pp. 184–195, 2016.

[21] F. Chao, J. Lv, D. Zhou, L. Yang, C.-M. Lin, C. Shang, and C. Zhou, "Generative adversarial nets in robotic chinese calligraphy," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1104–1110, 2018.

[22] R. Wu, W. Fang, F. Chao, X. Gao, C. Zhou, L. Yang, C.-M. Lin, and C. Shang, "Towards deep reinforcement learning based chinese calligraphy robot," *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 507–512, 2018.

[23] S. Xu, M. Tang, F. C.-M. Lau, and Y. Pan, "A solid model based virtual hairy brush," *Comput. Graph. Forum*, vol. 21, pp. 299–308, 2002.

[24] S. Xu, F. C. M. Lau, F. Tang, and Y. Pan, "Advanced design for a realistic virtual brush," *Comput. Graph. Forum*, vol. 22, pp. 533–542, 2003.

[25] S. Saito and M. Nakajima, "3d physics-based brush model for painting," in *SIGGRAPH Abstracts and Applications*, 1999.

[26] J. Lee, "B simulating oriental black-ink painting," in *B Simulating Oriental Black-ink Painting*, 1999.

[27] N. S.-H. Chu and C.-L. Tai, "An efficient brush model for physically-based 3d painting," *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.*, pp. 413–421, 2002.

[28] S. Strassmann, "Hairy brushes," in *SIGGRAPH*, 1986.

[29] H. T. F. Wong and H. H.-S. Ip, "Virtual brush: A model-based synthesis of chinese calligraphy," *Computers and Graphics*, vol. 24, pp. 99–113, 2000.

[30] W. V. Baxter and N. K. Govindaraju, "Simple data-driven modeling of brushes," in *SI3D*, 2010.

[31] C.-L. Liu, I.-J. Kim, and J. H. Kim, "Model-based stroke extraction and matching for handwritten chinese character recognition," *Pattern Recognition*, vol. 34, pp. 2339–2352, 2001.

[32] J. Zeng, W. Feng, L. Xie, and Z.-Q. Liu, "Cascade markov random fields for stroke extraction of chinese characters," *Inf. Sci.*, vol. 180, pp. 301–311, 2010.

[33] R. Cao and C. L. Tan, "A model of stroke extraction from chinese character images," *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 4, 368–371 vol.4, 2000.

[34] C. Lee and B. Wu, "A chinese-character-stroke-extraction algorithm based on contour information," *Pattern Recognition*, vol. 31, pp. 651–663, 1998.

[35] Y. Sun, H. Qian, and Y. Xu, "A geometric approach to stroke extraction for the Chinese calligraphy robot," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 3207–3212.

[36] F. Chao, Y. Huang, C.-M. Lin, L. Yang, H. Hu, and C. Zhou, "Use of automatic chinese character decomposition and human gestures for chinese calligraphy robots," *IEEE Transactions on Human-Machine Systems*, vol. 49, pp. 47–58, 2019.

[37] D. H. Gottlieb, M. Y. Hussaini, and S. A. Orszag, "Theory and applications of spectral methods," in *Theory and applications of spectral methods*, 1984.

[38] P. T. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," *2012 IEEE International Conference on Robotics and Automation*, pp. 2088–2095, 2012.

[39] *Makemeahanzi*, https://github.com/skishore/makemeahanzi, Accessed: 2019-09-10.

[40] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," in *Factor Graphs and GTSAM: A Hands-on Introduction*, 2012.

[41] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *Intl. J. of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[42] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, "Steap: Simultaneous trajectory estimation and planning," *Autonomous Robots*, pp. 1–20, 2018.

[43] E. Y. Y. Zhang and P. S.-P. Wang, "A parallel thinning algorithm with two-subiteration that generates one-pixel-wide skeletons," *Proceedings of 13th International Conference on Pattern Recognition*, vol. 4, 457–461 vol.4, 1996.

[44] K. W. Lo, K.-W. Kwok, S. M. Wong, and Y. Yam, "Brush footprint acquisition and preliminary analysis for chinese calligraphy using a robot drawing platform," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5183–5188, 2006.

[45] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *ArXiv*, vol. abs/1404.3785, 2014.

[46] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight : Standard platforms for service robot applications," in *Fetch and Freight : Standard Platforms for Service Robot Applications*, 2016.

[47] S. P. Boyd and L. Vandenberghe, "Convex optimization," *IEEE Transactions on Automatic Control*, vol. 51, pp. 1859–1859, 2006.

[48] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, pp. 1–139, 2017.